Your Mac as a Linux Box

A guide to open-source software on OS X

June 14, 2019

1 Background

In the late 90s, MacOS - the operating system used on Macintosh computers - had become a bloated, clumsy mess. Apple's solution was to turn to Unix-based technology developed by the NeXT company (founded by Steve Jobs during his "sabatical" from Apple). The result was the now widely used OS X, the most widely used operating system outside of Microsoft's Windows platforms. Because OS X is, at its core, a Unix system, it was quickly adopted by (amongst others) scientists who need both the open-source and command line capabilities of Unix/Linux systems, but also need access to popular point-and-click programs such as MS Office, MatLab, and others. OS X is therefore the "best-of-both-worlds" for scientists.

The move from a traditional Linux systems to OS X comes at a price, however. Apple insists on several non-standard implementations of languages (e.g., python), window managers, and file system layouts. This means that to unlock the full Unix/Linux-like capability of your Mac, you need to do a bit more setup work. This guide attempts to get you set up so that you can start coding like a Linux pro!

Note that this process will take *time*. Getting software, installing packages, and getting up-and-running requires plenty of time to download and compile libraries and packages. Make sure you're ready before you start.

2 Accessing the Terminal

Open up a Finder window and navigate to `Applications/Utilities/Terminal.app`. Start that up. Tadaa! That's all!

...I lied, it's never that easy. You can happily use the terminal and do most of the things you need to do, but I *strongly* recommend obtaining an X Windows System (or just X11 for short) compatable terminal. That is available via XQuartz, a project dedicated to creating a pure X11 experience on your Mac. Download and install the XQuartz terminal. Trust me. It's better. Once you have it installed, open a terminal and start playing.

A critical thing to understand in X11 is the copy-paste mechanics. In Linux, anything you highlight is copied to the clip board and can be pasted via a middle-click of the mouse. In OS X and XQuartz, **middle click of the mouse will paste any highlighted text in an X11 app OR text copied to the standard OS X clipboard, which ever happened last.** If you do not have a middle button on your mouse, an equivalent input is to hold the option key and left-click. If this behavior is not working, be sure to check "Emulate three button mouse" in XQuartz's preferences.

This mechanism seems clunky at first, but is much faster than the usual command-C command-V keystrokes. Think about it: you usually need to put your hand on the mouse, highlight text, then move your hands to the keyboard to press command-C to copy it. Then, back to the mouse to select where you want the text to be placed, then back to the keyboard for command-V paste. With the middle-click paste, your hands stay on the mouse the whole time.

An important trick is that you can customize what your terminal window looks like. Under the "Applications" menu in XQuartz, click on "Customize...". You will see the default "Terminal" command, which simply calls xterm. If you edit the "command" column of an existing row, or create a new row and edit the command column of that, you can pass options to the xterm command that changes the font, color, and other behavior to your terminal. For example,

```
xterm -fa Monaco -fs 16 -geometry 60x13
```

creates a window that is 60 by 13 characters in size while changing the font and font size for better clarity. If you click "Add Item", you can create a new command for more customization. Suppose you want a window that automatically logs into a certain machine and sets new colors so you know that window is different than your default. Do this:

```
xterm -fg Gold -bg DarkBlue -T Umich -e ssh -YC yourname@login.itd.umich.edu
```

Now, when you select this option from the "Applications" menu, you'll get a window that automatically logs into your Umich webspace. For more options, see the man page for the xterm command.

3 Text Editors

At this point it is critical that you have a go-to text editor within your X11 environment. The two most popular are Emacs and Vi. They are also both already installed on all Linux-like machines, including your Mac. I'm not going to go in to which you should use or shouldn't use, or the advantages of each. However, I prefer Emacs and find it to be very powerful. An Emacs command cheat-sheet can be found here. At this point, pick either Emacs or Vi and do a quick internet search on how to start, edit text, save a file, and exit. Be sure you can do this before moving forward.

Let's assume that you will be using Emacs. Start an Emacs session by typing "Emacs" at the command line prompt. The terminal then enters Emacs mode, and you can begin typing in a blank text file. If you want to save what you're typing, press C-x C-s (that's control and x at the same time, then control and s at the same time.) Emacs will ask you to name the file. To quit Emacs, press C-x C-c. At this moment, the version of Emacs you are using has no point-and-click interfaces (called Graphical User Interfaces, or *GUIs*). In Section 6, we'll see how to rectify this.

4 Editing the Configuration File

An important and unavoidable part of customizing *any* Linux-like environment is editing the configuration files. You'll find these in your home directory (type `cd` to get there) by typing `ls -a`. Most of those files that start with a dot are config files. Look for `.profile`, `.bashrc`, or `.cshrc`. The first two are for users of BASH, the Bourne-Again Shell. The latter is for C-Shell users. What one you edit depends on what shell you are currently using. To figure that out, type `echo $0`. **This guide assumes you are using BASH, as it is the default for modern OS X and Linux machines.**

The shell configuration files are a series of commands that are run when you start a new terminal. You can use these to set your search path, set aliases, configure code repositories, and more. You'll

be constantly updating your shell config files as you expand your software library and customizing your computing environment.

Let's do a few things to customize the shell to your preferences. Let's start by finding what configuration files you have. Use `ls -a` in your home directory to see if any of the following files exist:

- `.profile`
- `.bashrc`
- `.bash_profile`

The difference between these three is subtle and sometimes system dependent. Different configuration files are used by Bash in different situations, so you may find that one works and others appear to have no effect. It appears that, for OS X, the file that seems to be the most robust is `.profile`.

Open one of the above files using emacs (e.g., `emacs .profile`. If none of the above exist, using `emacs .profile` will create that file. Move your cursor towards the end of the file, and add this text:

```
# Set default text editor:
export EDITOR='emacs -nw'
```

Note the comment, preceeded by a hastag, so that you can remember what this line does in the future. If you're a Vi user, replace `'emacs -nw'` with `'vi'`. This command changes the default text editor that will open for programs such as Visudo.

4.1 Search Paths

The next thing you'll want to do is change your *search path*. This is the set of directories that your computer searches everytime you enter a command. When you type `ls` to look at the contents of the current directory, BASH is quietly searching for an executable program called "ls". If you want to know where BASH finds this command, use `which` at the shell prompt:

```
which ls
```

This should return `/bin/ls`. To see your search path, type the following at the shell command prompt:

```
echo $PATH
```

You should now see a colon-separated list of directory paths. Let's edit your config file to change your search path. Suppose you have a directory full of scripts that you would like to access from any location. Make this directory now by typing `mkdir myscripts` from your home directory. Now, add these lines to your config file:

```
# Add scripts directory to search path
export PATH=$PATH:~/myscripts
```

..where ~is shorthand for your home directory path. This line keeps the default search path in tact and appends a new entry to it. The shell will always search the path list in order, so `/myscripts` will be searched last. If there are multiple files with the same name in different searched files, the first one found will be used. Therefore, if you want your script directory to be searched first, simply change the order like so:

```
# Add scripts directory to beginning of search path
export PATH=~/myscripts:$PATH
```

4.2 Activating & Checking Config Files

Now, let's get BASH to recognize the changes you've made. To make these changes take effect immediately, type `source ~/.profile` or `source ~/.bashrc`. Otherwise, they will take effect once you open the next terminal window.

Test to make sure that these changes by opening a new terminal and using these commands at the prompt:

```
echo $SHELL
echo $EDITOR
```

The first line should return your colon-separated list of directories; make sure that your new directory `~/myscripts` is in that list. The second line should print out your default text editor. Make sure it lists the one you set.

If these changes do not seem to be taking effect, there are several things that could be wrong. If you're seeing an error message when you open a new window, that means that you've made a mistake in your config file. One-by-one, comment out the lines you added and open a new terminal between each change to find out exactly what line is giving you troubles. Try entering that line into the command line prompt directly and change it to figure out how to get it to work. If there is no error but the config file changes do not seem to have any effect, it is likely that your shell is not using the configuration file you are chosing to edit. Use the `echo` command in each of your files to cause them to print messages to the screen:

```
echo 'This is your bash_profile file checking in!'
```

Of course, customize the message to match each file's name. Now, open a new terminal window to see what config file is being used and move your commands to that one. Finally, if none of the above items work, turn to Google. Be sure you know what shell you are using (`echo $0`) as you search.

5 Sudo Access

Installing software requires administrator access. You know this already: when you install new software on your Mac, it asks for your admin user and password. At the command line, you do this via `sudo`, which stands for *super user do*. Any command preceeded by `sudo` at the terminal prompt will be executed with administrator rights. This is required for installing new software from the command line prompt.

Only some users can use `sudo`, and you will need it moving forward. To get sudoer access, you must change the *sudoer's file*. The easy way to do this is first log into OS X using an administrator's account. Then, from the command line, use the `visudo` command:

```
export EDITOR='emacs -nw'
sudo visudo
```

Enter your administrator password when asekd. This will open the sudoer's file for editing in your preferred editor (set by the `EDITOR` environment variable. Note that because we are now using a different account (i.e., your admin account), we have to set this variable again! Find the lines inside of the sudoer's file that look like this:

```
root    ALL=(ALL) ALL
```

Copy that line, but replace 'root' with your regular user name. **Be sure to leave the root line intact or you will have big problems!** The result should look something like this:

```
root    ALL=(ALL) ALL
dwelling    ALL=(ALL) ALL
```

Save and exit your text editor and return to your normal computer account. Test your sudo access:

```
sudo echo "it works!"
```

Sudo will ask for your password and then, if everything was done correctly, print out your message to screen.

6 Installing MacPorts

Open-source software is the backbone of any linux-based system. Getting new software isn't done like other platforms, where you download installers or use an app store. Rather, you depend on a *package manager*. A package manager allows you to search available open-source software, automatically download a program you want to install and all of its dependencies, and install everything you need.

There are options for OS X. The three big ones are, in order of age, Fink, MacPorts, and Homebrew. I recommend MacPorts because it is well established and the packages are well maintained. The following guide is a crash-course to installing and using MacPorts. For a more complete description, check out the official guide and don't forget to GDS when you get stuck.

6.1 Installing and Configuring MacPorts

Start by downloading and installing MacPorts from the project website. Follow the instructions on the website, including installing XCode from Apple. Part of the installation process is editing your search path to include `/opt/local/bin`, where MacPorts keeps software. Be sure to check that your shell config file was correctly edited by the installer. A recommended change is to put the MacPorts directory at the beginning of your search path, like so:

```
export PATH=/opt/local/bin/:$PATH
```

This prevents your shell from finding Mac's default programs before your shiny new MacPorts versions.

6.2 Installing Code

Let's use MacPorts. Note that there are more in-depth guides to doing this, such as the one provided on the MacPorts website. Start by finding a package you want to install, such as emacs. To list ALL available software, use this command: `port list`. That's a long list! let's search for just emacs:

```
port list | grep -i emacs
```

This will print out all versions of emacs that are available. The plain-old `emacs` seems good enough. For each sotware package, there are also *variants*, which allow customizations of your install. We can see our variants for emacs like so:

```
port variant emacs
```

The `x11` variant gives us emacs in its own window, which is what I prefer! So finally, let's install it:

```
sudo port install emacs +x11
```

...and MacPorts goes to work. This will take a while, but it does it eventually. If anything ever goes wrong, find the exact error message and search it online. The MacPorts user community is very large and active. Very rarely is there a problem that you have that someone else hasn't already solved.

6.3 Installing Python

Let's see how we install Python and all of its critical science packages via MacPorts:

```
sudo port install python37
sudo port select --set python python37
sudo port install py37-ipython py37-scipy py37-numpy py37-matplotlib ipython_select
sudo port select --set ipython py37-ipython
```

The `select` commands make it so that we can call python and ipython using their names, not their names and their versions (i.e., `ipython-3.7`).

6.4 Other Useful Packages

Here's a quick list of some tools that are very useful. They are listed by their MacPorts name and suggested variants.

- `tkdiff`: a gui-based tool for comparing two text files.
- `texlive`: Installs LaTeXand some basic libraries. Most LaTeXadd-on libraries can be installed via MacPorts, too.
- `TexShop`: a gui-based tool for editing LaTeXfiles. A clickable executable file will be placed in your Applications/MacPorts directory.

- `cdf`: NASA's Common Data Format library.
- `gcc49`: The Gnu C Compiler. Includes gfortran, the leading open-sourced Fortran compiler.
- `openmpi`: Open source Message Passing Interface library for compiling parallized software.
- `ImageMagick +x11`: Powerful open-source image manipulation tools.

7 Confirming Your Path

At this point, you're nearly done, but it's a great idea to make sure that your newly installed software is properly accessible. Remember that OS X has its own non-standard versions of programs such as Python, so even though you're using python, it may not be the right one, and only the right one will have your extra installed goodies, such as Numpy. Use `which` to ensure that you set your software path correctly.

```
which python
```

If you see anything besides `/opt/local/bin/python` here, be sure to revisit setting up your search path in Sections 4.1 and 6.3. Check for programs with a *similar* name using tab-complete (i.e., typing `python` then pressing tab repeatedly to see if there are other versions of python.) Use `which` on those programs. Are they in the right place? If so, use MacPorts' select syntax to set the default version of python, as we did in Section 6.3.

8 Epilogue

...and you're off! Hopefully this primer has set you forward successfully. Working from the command line is immensely powerful and becomes second nature the more you use it. It's not without its issues, though, so be ready for more challenges. In the end, it is an excellent way to get science done as efficiently as possible.

For those who have never used a pure Linux machine, I urge you to find an excuse to do so. The steps outlined above are obnoxious enough to make the novice user question the wisdom of turning to a command line approach. However, setting up a real Linux system is so trivial and fast that it makes you wonder why Apple doesn't include the command line tools, including package managers, in a more native fashion. On a system that fully embraces the open-source approach, the whole process is elegant and fast. Indeed, many people who start with Macs wind up with a dedicated Linux machine on the side.